AN OPTIMISED NAÏVE-BAYES DETECTION SYSTEM

By

TONI KUNNEL

Bachelor of Engineering in Computer Science

Bharathiyar University

Coimbatore, India

2001

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2005

AN OPTIMISED NAÏVE-BAYES DETECTION SYSTEM

Thesis Approved:

Dr. JOHNSON P THOMAS

Thesis Advisor

Dr. NOHPILL PARK

Dr. DEBAO CHEN

Dr. GORDON EMSLIE

Dean of the Graduate College

ACKNOWLEDGEMENTS

At this point I would like to thank God Almighty for guiding me in the right direction and rewarding my efforts. In fact I am grateful towards my advisor Dr. Johnson Thomas for his guidance and valuable ideas. I am really glad for all the good work that he does with the students. My heartfelt thanks to my committee members, Dr. Nohpill Park and Dr. Debao Chen, for providing their opinions and suggestions.

Also I would like to extend my gratitude to Dr. John Chandler, Dr. George Hedrick and all other computer science department faculty & staff for their assistance in times of need. My work would not have been completed with out the guidance of Dr. Zhigang Zhang, Department of Statistics. I thank Oklahoma State University for providing all the facilities needed for the research. I would like to express my sincere thanks to the System Managers and the Information Technology Division Team for their timely cooperation. I thank all my friends here at Oklahoma State University for their support and encouragement when I needed them the most

I am ever thankful to my parents and my family (Kunnel & Panachoor) for their continuous support. Thank you for all your love and encouragement that will take me through every obstacle.

TABLE OF CONTENTS

Chapter **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER I

INTRODUCTION

1.1 INTRUSION DETECTION

Intrusion Detection is now a significant component of any security system. The purpose of an intrusion detection system (or IDS) is to detect unauthorized access or misuse against a computer system. Intrusion detection systems serve as burglar alarms for computer systems and networks. They sound alarms and sometimes even take corrective action when an intruder or abuser is detected. If a Masquerader or Hacker can gain access and use your internet access, then they can use your machine to launch other attacks on other computers while keeping themselves well hidden. For example, there are certain applications that take days to months to run a series of processes on even the fastest computer. If a Masquerader can gain access to 1000 computers and utilize their combined processing power, a process that would take a month on a single computer could complete the operation in less than an hour.

These Intruders or Masqueraders could be anyone, they could be kids who use their parent's company account in spite of the company policy or people just playing jokes on other users or they could be malicious intruders intentionally trying to hide their identity by impersonating other users, such types of masqueraders are considered to be Insiders.

There are some intruders from the outside who immediately try to gain access through a valid user and they can be very dangerous. According to a survey, most of the financial damages were caused by the Insider and not by the Outsider [8]. Moreover, it was estimated that about one thirds of the total damages were caused by the Insider.

1.2 MASQUERADE DETECTION

A Masquerader or an Intruder behaves way different from the Proper User. Most of the well known methods in Computer Intrusion Detection fall under two broad categories and they are 1) Pattern Recognition or Misuse and 2) Anomaly Intrusion Detection. The art of recognizing intrusion patterns from the previously observed intrusions is known as pattern recognition. Most of the researchers consider it to be the first line of defense. Apparently it is the most powerful technique when the intrusion method is known. Illegal User behavior is generally difficult to comprehend because they can be a lot of ways in which an Illegal User could intrude. Moreover, Hackers or Masqueraders come up with novel strategies of attack and hence pattern recognition would not be very effective in stopping the intrusion.

A proper User behavior can always be recorded and any behavior that deviates from the proper behavior is considered to be Intrusive. Hence this type of Detection is named as Anomaly Detection System. Anomaly detection techniques can be very useful with novel attacks and in fact, these techniques need some amount of statistical information to support themselves. In Anomaly detection techniques, a profile of the valid user is built and any behavior that does deviate from the profile is considered to be intrusive. Hence Anomaly Detection is the most appropriate method of detection. Some of the categories of intrusion attacks are eavesdropping and packet sniffing (passive interception of network traffic), snooping and downloading, tampering or data diddling (unauthorized changes to data or records), spoofing (impersonating other users, e.g. by forging the originating email address, or by gaining password access), jamming or flooding (overwhelming a system's resources, e.g. by an email flood), injecting malicious code (via floppy disks, email attachments, etc.), exploiting design or implementation flaws (often buffer overflows; overflows overwrite other data and can be used to get control over a system) and, cracking passwords and keys.

Previous approaches have used the dataset studied and presented by Schonlau [2]. This dataset had only on parametric field in it namely the "command". In this thesis, we propose a novel approach to masquerader detection based on command and CPU Time. Following this introduction, is Section II that explains the Users dataset used in our thesis and the UNIX commands used to generate the dataset. Section III gives a brief overview of different methods used by previous researchers for Masquerader Detection System. Section IV discusses the Online and the Offline Naïve Bayes Classifier and our methodology to increase the accuracy of the system. Section V summarizes the main conclusions of this research as well as potential for future work.

CHAPTER II

LITERATUR REVIEW

2.1 DEFINITIONS

2.1.1 Hit rate

The Naïve Bayes Algorithm is a learning technique which detects Masqueraders having an Illegal entry to the system. Hit rate is the rate at which the Algorithm correctly identifies proper users and Masqueraders.

2.1.2 False Alarm Rate

False Alarm Rate, in other words, would be how incorrectly the algorithm has detected. There might be a lot of alarms even with proper or valid user.

2.1.3 Missing Alarm rate

Sometimes the Naïve Bayes Algorithm would not produce a good accuracy rate and hence it might not produce an alarm for a masquerader. This detection rate is called the Missing-Alarm-Rate.

2.2 OVERVIEW OF INTRUSION DETECTION SYSTEM

Simulating real-time IDS (Intrusion Detection System) is a huge task and we require a detailed study of different components involved. Many researchers have been trying to study these components and have succeeded in reducing the High False Alarm Rate. Schonlau [2] and his colleagues presented the dataset in full detail and reviewed six techniques to analyze User command log files. They reported a high of 70% hit rate and 6.7% false alarm rate. Maxion and Townsend [1] later achieved better results by using Naïve Bayes classifier which recorded a hit rate of 61% and a false alarm rate of 1.3%. Also Yung [7], proposed a Self-consistent Naïve Bayes classifier which resulted in a low missing alarm rate of 40% and a false alarm rate of 1.3%.

2.3 PREVIOUS RESEARCH

Schonlau [2] used keyboard command data from 50 users and these fifty user datasets were injected with data from outside the community of 50. Hence the data from outside is considered to be the intrusive data. Each user data file consisted of 15,000 commands, within which the first 5000 were considered to be the Truth or considered as legal commands while the rest of the commands have been formed with the injection of commands from users outside the community of 50. Their objective was to find if each block of 100 commands (which is called a session) was typed by a Proper User or a Masquerader.

2.3.1 DATA CONFIGURATION

The data were configured in two ways [1]: (1) randomly injected with data from users outside the community of 50 (which approximate an incursion by a masquerader) and (2) each user crossed with every other user to compare the effects of every user acting as a “masquerader” against all other users. The first configuration is that employed by Schnonlau, hereafter referred to as the SEA configuration. Some of the contrasts between the first and the second configuration are (1) the first configuration provides a consistent set of intrusions against all users, which allows meaningful error analysis. (2) The second configuration samples a greater range of intrusive behavior.

2.3.2 COMPARISON RESULTS FOR SEVERAL DETECTION SCHEMES

Six masquerade detection schemes were used on the dataset packed with “Masqueraders” and their results were compared in the review paper by Schonlau [2]. The investigators targeted a false-alarm rate of 1%. Most of the methods hit low hit rates (ranging from 39.4% to 69.3%) and high false alarm rates (ranging from 1.4% to 6.7%). The results were compared using cluster analysis and ROC curves. Their results are shown graphically in Figure 1.

- Bayes 1-Step Markov.

  A single step transition from one command to the next [3] is the logic behind this detector. The observed transition probabilities should be consistent with the historical probabilities and this determined by the detector. The technique worked really well in terms of correct detections but failed badly with respect to the desired false alarm rate.

- Hybrid Multi-Step Markov.

  This technique is based on Markov chains and is due to Ju and Vardi [4]. The implementation of this model is toggled between a Markov model and a simple independence model, depending on the proportion of test data that was not observed in the training data. This method peaked highest in terms of performance among the methods tested.

- IPAM.

  This detector like the Bayes 1-step Markov method is based on single-step command transition probabilities which are estimated from the training data. The Incremental Probabilistic Action Modeling (IPAM) was developed by Davison and Hrish [1.] for their work in predicting sequences of user actions. The result of this method is depicted in Figure 1 where it is compared with other methods. IPAM's performances have not been satisfactory.

- Uniqueness.

  This method was developed by Schonlau and Theus [13]. Commands that are not seen in the training set may indicate a masquerade attempt. In fact, the less frequent a command is used by the Users, the more it is intrusive. Uniqueness was relatively poor in performance when compared to other methods but it was the only method that approached the target false alarm rate of 1% [2].

- Sequence-Match

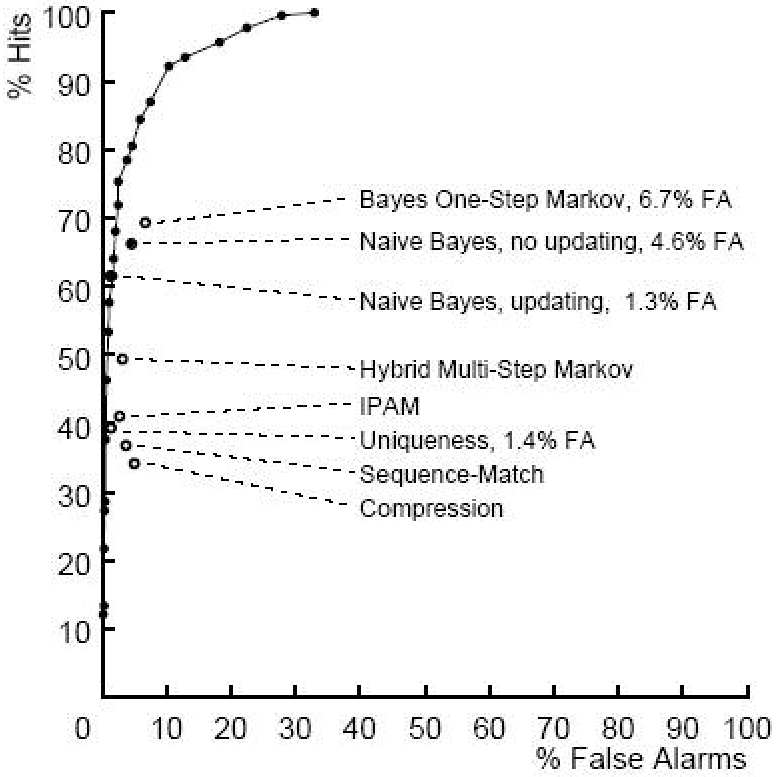
  The method was developed by Lane and Brodley [5] and the results have been portrayed in figure 1. This method computes a similarity match between the most recently used commands and the user profile. With Schnlau's dataset it was poor performer.

- Compression

  This method was built under the belief that new data would compress at the same ratio as the old data from the same user, and that data from a masquerading use will compress at a different ratio, and hence we could distinguish between the proper and the masquerader. This idea was developed by Schnalau and karr [2] and the comparison results are shown below

Comparison of results of different Detection Schemes

Figure 1. Compression was the worst performer of all these methods [2].



We now discuss the comparison results of different intrusion detection techniques as shown in the above figure. The curve or the arc formed by the X-axis and the Y-axis is called the ROC curve (Relative Operating Characteristic curve). The X-axis represents the percentage for the number of Hits and the Y-axis represents the percentage for the number of false alarms.

2.4 SPLICING OF SESSIONS

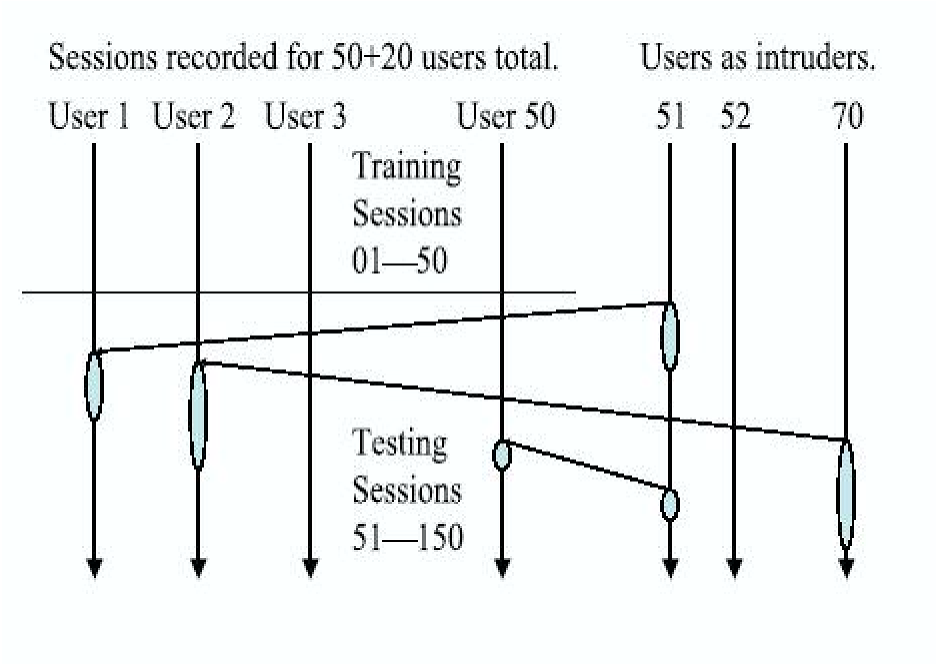


Figure 2 [1] Splicing of sessions.

The above figure shows how splicing is done with the datasets. Each user has a dataset of 15,000 commands and in turn we split the commands into sessions of 100 each. Hence we have 1 – 50 sessions in the training set and 51 – 150 sessions in the Testing Data. Furthermore we have a couple more users which are classified as masqueraders and they do not belong to the community of the 50 users (51 - 70). They choose a couple of test sessions from these Masqueraders and intersperse within the test data with a certain probability. This entire process is called splicing of sessions. There are certain rules that we need to follow while we do the splicing of sessions.

The EM algorithm is used to assign probabilities to the newly unidentified sessions from the test session data [7]. There is a good reason for using the EM algorithm. We do not want the classifier to classify each session in a greedy way rather we would like to classify it after enough sessions for a good classification has been recorded. Basically, we apply the log-likelihood function to the existing probabilistic model. Conclusively our aim would be to maximize the log-likelihood function in the presence of incomplete data. The theory behind our approach and a brief summary of the Bag-of-words model is discussed below.

2.5 BAYESIAN APPROACH

2.5.1 CLASSIFICATION THEORY

Examples of normal behavior of users are available in plenty in different datasets but the intrusive behavior is hard to collect. This is because these kinds of datasets are very confidential or hard to capture. Similarly, in relation to masquerading sessions, many examples of proper sessions are available. Masquerading sessions in the training data are not known and moreover, masquerading sessions in the test data are expected to be rare.

Traditionally, all the proper sessions in the training data are collected to form the proper User model. The proper sessions of all other users from the training data are taken to form the Masquerader model for the current user. This approach would be quiet effective with our dataset since all the masquerading sessions have been formed from the proper sessions of other users. Further, all new test sessions are compared against the profiles for the proper user model and the masquerader model. The anomaly detection problem has been transformed into a classification problem over the Proper user class and the artificially created Masquerader class. This transformation has allowed many powerful techniques from the Classification theory to be applied on the anomaly detection problem.

Even in a more general context of anomaly detection, the profile of the proper sessions is defined not only by the proper sessions, but also by the sessions of other users [3]. The extent of the proper profile in feature space is most naturally defined through both the proper cases and the intruder cases [3]. From the above, we see how beneficial this transformation for the anomaly detection problems is.

2.5.2 BAG-OF-WORDS MODEL

The Bag of words model is one of the most widely used model in Information Retrieval (IR). The actual Text document is broken down into bag of words and we just keep track of the occurrences of each word rather preserving the sequence of the words. In other words we completely ignore the sequence information of the Text document. For classification of text documents, this simple model often performs better than more complicated models involving sequence information. Let C = {1, 2… n} be set of all possible commands and 'n' is the last command in the set. In reality, all the commands that form the text document are broken into small groups of equal length called sessions.

Hence a session number‘s’ is a finite sequence C_s = (C_{s1}, C_{s2}, C_{s3}, ….., C_{szs}) of commands. In other words C_s is a sequence of commands with length ‘zs’.

As demonstrated in [1], the bag of words model outperforms many more complicated models attempting to take advantage of the sequence information. In particular, the techniques reviewed in [2] and earlier techniques based on a Markov model of command sequences achieved worse results.

2.5.3 GENERAL THEORY

We have examined the log files of different user accounts at the Solaris Computer Lab of Oklahoma State University. These log files were collected from different user accounts logged on to the a.cs.okstate.edu server. Each Log file consists of different records based on the attributes that are fed as input to the system. In fact, every User has his own unique style of expressing or talking to the system. Different attributes of the log file can be considered as different levels of monitoring a system. Previous researchers have considered only “COMMAND” level. In our work we would develop a combination of the online and offline classifier with a trust node and consider the CPU time as another parameter which increases the level of surveillance. Some of the parameters available are Command, CPU time, User ID, Date & time of login. Later in this paper we define each of these fields. Moreover, each record or entry in the Log file represents or contributes to a part of the complete behavior for that User.

Each parameter or field in our dataset can be looked up on as different levels of monitoring. The first level of monitoring is the Command level and this can be extended to different levels of monitoring by considering the rest of the attributes. Each User has a unique role or behavior to play in an organization. Therefore he would have access to only a certain type of commands. This belief helps us to visualize a unique pattern for every proper user in the Organization. The interloper or the “improper user commands” may comprise read or write access to private data, acquisition of System privileges, installation of malicious software etc. Hence the pattern produced by the improper user is way different from that of a proper user. To detect these intruders, behaviors of proper user are recorded and any deviant behaviors are considered as intrusions. An anomaly-based intrusion detection technique is followed in this approach. Some of the common types of UNIX commands used in our data set are,

- File Management.
- Comparison and searching commands.
- Text processing.
- Shell and other programming.
- Communications.
- Storage commands.
- System status.
- Miscellaneous commands.

CHAPTER III

DATASET

3.1 DESCRIPTION

The dataset was prepared from the log files of different users that logged in to the Computer Science Administrative Server (a.cs.okstate.edu). Most of the UNIX commands and some additional fields were captured into a dataset. We used the “lastcomm” command to generate the data and later ran a script file. The syntax used to generate the list of a particular commands typed by the user is <lastcomm> <userid>. This gives the commands in the reverse order and the file is updated at 2AM every day. Well, the server running on the UNIX environment generates an accounting file or the log file for different users logged each day. In reality, the “lastcomm” command access these log files of different users to generate the data for different users. The command gives us information on previously executed commands. And with no arguments the “lastcomm” displays information about all the commands recorded during the current accounting file’s life time. If called with arguments, “lastcomm” displays accounting entries with a matching command-name, user-name, or terminal name.

3.2 FEATURES OF **”lastcomm”** COMMAND

For each process entry, “lastcomm” displays the following items of information.

- The command name under which the process was created.
- One or more flags indicating special information about the process. The flags have the following meanings:

  F - The process performed a fork but not an exec.

  S - The process ran as a set-user-id program.
- The name of the User who ran the process.
- The terminal which the User was logged in at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process exited.

CHAPTER IV

METHODOLOGY

4.1 BAYESIAN CLASSIFICATION

Bayesian Classifiers are developed to predict class membership probabilities, such as the probability that a given sample belongs to a particular class. Previous studies have found a simple Bayesian classifier known as the naïve Bayesian classifier to be comparable in performance with decision tree and neural network classifiers [11]. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases. Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence. Surprisingly, good performance is exhibited in many domains that contain clear attribute dependences [8].

4.1.1 BAYES THEOREM

Let X be a data sample whose class label is unknown. Let H be some hypothesis, such as the sample X belongs to some type of class C. Here we have to determine P (H|X), the probability that the hypothesis H holds true for the data sample X. In other words P (H|X) can be defined as the posterior probability, of H conditioned on X. P(H) is called the Prior probability of H. The posterior probability, P (H|X), depends on additional knowledge but the prior probability P (H) is independent of data sample X. Similarly we could define P (X|H) as X conditioned on H.

We use the Bayes theorem to calculate these probabilities.

P (H|X) = P (X|H) P (H) / P(X)

4.1.2 NAÏVE BAYESIAN CLASSIFICATION

The Naïve Bayesian classifier or the Simple Bayesian classifier works as follows

1) The data sample X is represented by a n-dimensional vector X = (x_1, x_2… x_n) and this has been contributed by n different attributes such as A_1, A_2… A_n.
2) Let us consider that there are m classes, C_1, C_2… C_m. We also have a given Sample, X, for which the Naïve Bayes classifier would assign the unknown Sample X to the class C_i if and only if

   $P (C_i \mid X) > P (C_j \mid X)$ for $1 \leq j \leq m$, $j \neq i$.

   This is known as maximum posterior hypothesis. Hence by Bayes Theorem,

   $P (C_i \mid X) = P (X \mid C_i) P (C_i) / P (X)$
3) Apparently P(X) is a constant value for all classes and hence we need to maximize $P (X \mid C_i) P (C_i)$. Now if the prior probabilities of the classes are not known we assume that prior probabilities are equally likely, such as $P (C_1) = P (C_2) \ldots = P (C_m)$ and hence we need to maximize $P (X \mid C_i)$. Otherwise we maximize P (X |

C_i) P (C_i). We estimate P(C_i) = T_i / T, where T_i is the total number of training samples of class Ci and T is the total number of training samples.

4) It would be extremely computationally expensive to compute P(X | C_i) when we have datasets with many attributes. In order to reduce this complexity, we make the assumption of class conditional independence. We mean that the values of the attribute are conditionally independent of one another, given the class label. Hence,

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

The probabilities P (x_1 | C_i), P (x_2 | C_i)… P (x_n | C_i) can be estimated from the training samples.

P (x_k | C_i) = T_{ik} / T_i, where T_{ik} is the number of training samples of class C_i having the value xk for A_k and T_i is the number of training samples belonging to C_i.

5) In order to classify an unknown sample X, P (X | C_i) P (C_i) is evaluated for each Class C_i. Sample X is then assigned to the class C_i if and only if

$$P(X \mid C_i)\, P(C_i) > P(X \mid C_j)\, P(C_j) \quad \text{for } 1 \le j \le m,\ j \neq i.$$

The objective of the Thesis is:

(1) To simulate an Online and an Offline Bayesian classifier which toggles around an optimal Trust Value.

(2) Increasing the accuracy of the Probabilistic Model (Online and Offline Classifier)

by the addition of a parameter (“CPU time”) and hence increasing the level of monitoring.

Previous Research

(1) In the past, they have used only one type of classifier and here we have a combination of two classifiers which work simultaneously on the dataset.

(2) Previous Researchers have used a different dataset with only one field (i.e.) “Command” and arrived at an Accuracy Rate. But in My research, I have prepared dataset with different fields such as “Commands”, “CPU Time”, “Date and Time of Login”, “UserID” but the level of monitoring would be through the commands and the CPU Time.

4.2 PROPOSED APPROACH

This section would be followed by a brief description of the dataset and our proposed algorithm in detail. The simple Naïve-Bayes classifier is perhaps the most widely studied classifier and is the Bayes-rule classifier for the bag-of-words model. By Bayes inversion formula, the posterior probability $P(u|c_s)$ of user ‘u’ given the sequence c_s is,

$$\mathbf{P(T) = P(u|c_s) * P(u|cput)}$$

Here P(T) is the Total probability which includes both the command and the CPU Time. ‘cput’ is the CPU Time and also,

$\mathbf{P(u|c_s) = P(c_s |u) P(u) / P(c_s)}$ which is directly proportion $P(c_s |u) P(u)$.

$$P(c_s |u) = \prod^{z_s} P(c_{sk} |u) = \prod^{C} p^{nsc}$$

k=1 k=1 uc

where n_{sc} is the total count of command c in session s.

Table 1: Dataset for User 1.

1st 5000 Commands and time from the training set

Next 10000 commands from the Test set.

	COMMAND	CPU time (secs)
1	Procmail	0.02
	⋮	
4999	biffmsg	0.08
5000	spamprob	0.06
	⋮	
10000	Lynx	0.04
	⋮	
15000	scan	0.02

Brief Overview of the Dataset:

Data sets for 10 users were created and each User had 15,000 records in it. A sample framework of the Dataset for user 1 is shown in **Figure 1**. Every User Dataset has the same structure as above. Let us now consider the dataset for USER 1. We now build 2 Models from the Training Set, one is the proper User model and the other the improper User or the Masquerader. The Proper User Model is build from the training set of USER 1 which consists of 5000 **Records.** The Masquerader Model is built by taking 550 commands each from the training dataset of USER2 – USER10. Hence (9 * 550 = (approx) 5000).

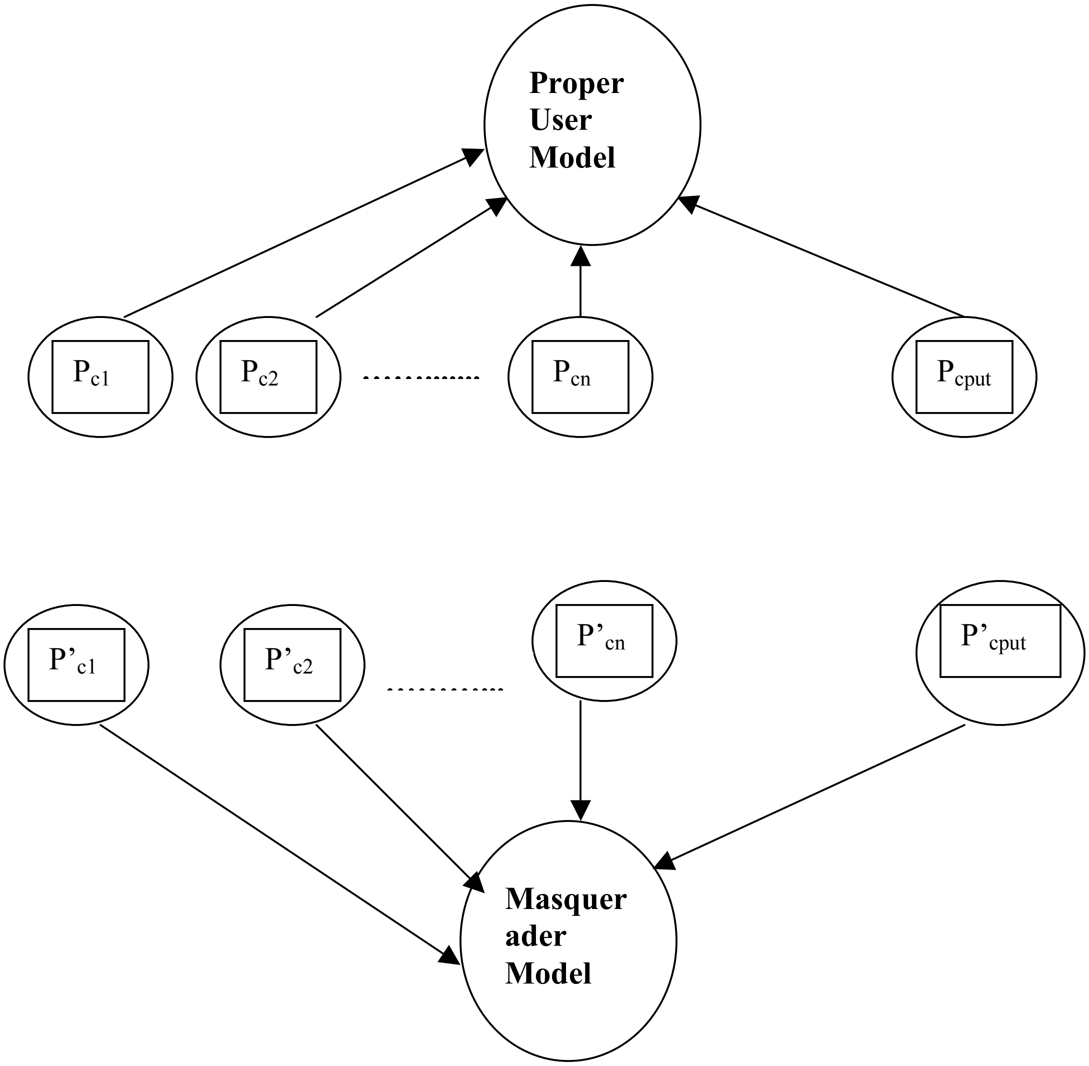
So now we have the proper User model and the Masquerader Model for USER1. Also, we need to test this Model by inputting the Test Dataset for USER1.

		COMMAND	CPU Time
1st session which consists of 100 records	5001	procmail	0.02
		⋮	
	5100	biffmsg	0.08
⋮			
50th session consisting of 100 records	10001	formail	0.01
		⋮	
	10100	ksh	0.02
⋮		⋮	
100th session which consists of 100 records	14901	comp	0.01
		⋮	
	15000	more	0.01

Table 2: Test Session for User 1.

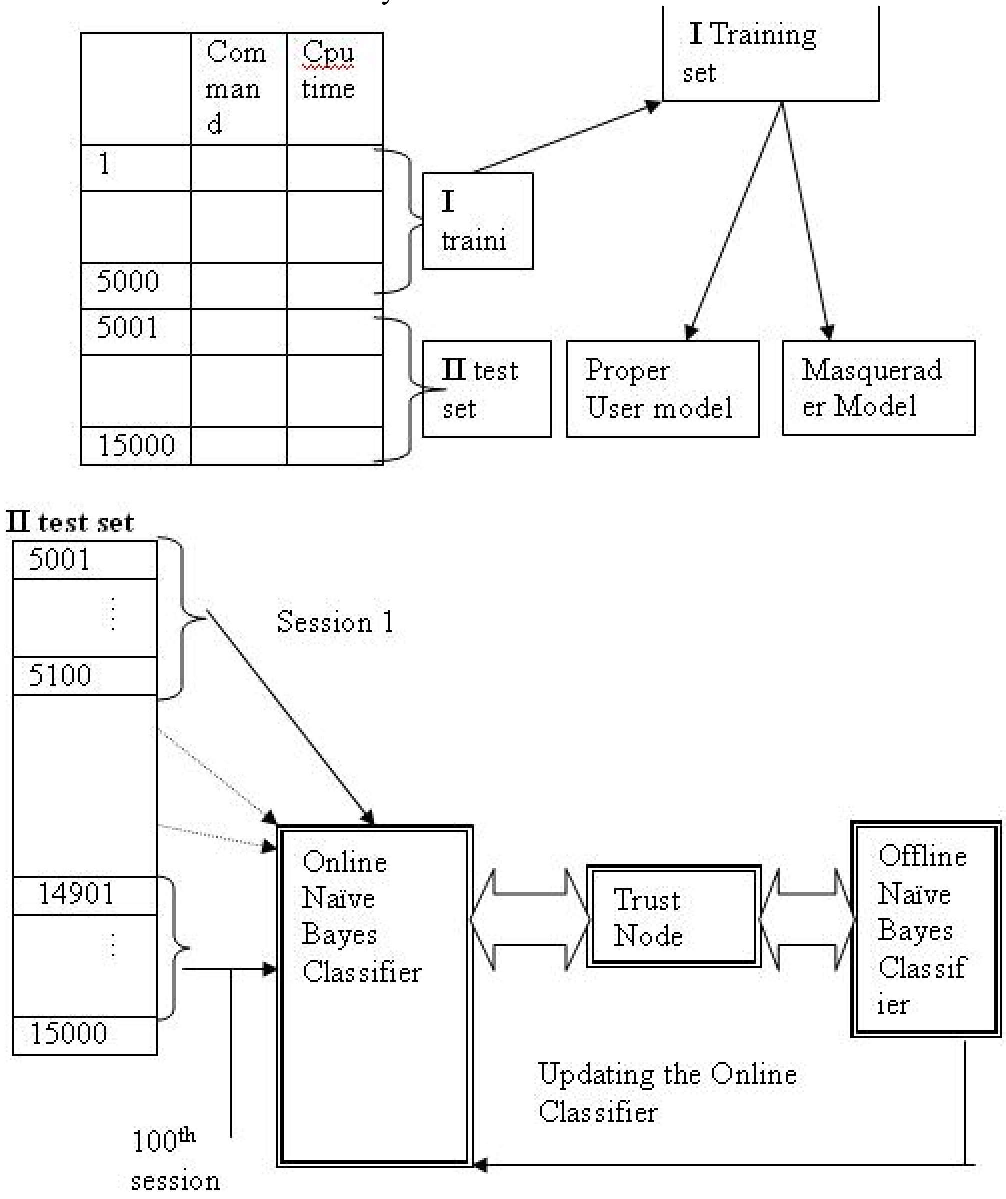
As seen from the Figure 2, the test session of USER1 which starts from 5001 – 15,000 is divided into 100 test sessions (sub sessions, to be precise). Each test session has 100 records in it.

4.2.1 NAÏVE BAYESIAN NETWORK



2.2 ONLINE & OFFLINE NAÏVE-BAYES DETECTOR

Figure 4. Online & offline naïve-bayes detector



Let us now see how the Bayesian Detector works. The detector consists of an Online and an Offline classifier. Moreover, there is a trust node (it stores the trust value) in it. An over view of this detector can been seen in Figure 4.

Now we have the first Test Session (Test Session 1- which consists of 100 records) coming to the Bayesian Network Detector. The Online and the Offline Classifier would start to work on the Test Session 1 simultaneously.

4.2.3 TRUST MODEL

The Total probability is the product of the probabilities of both the commands and the CPU time. Also, the probability of the time parameter can be considered as a weighted probability with the commands to give the total probability. The Online Classifier builds a Model and immediately classifies Test Session 1 as a Proper or a Masquerader Session. Meanwhile, the Trust Node calculates the trust value.

Note:

The word trust value is different from Threshold Trust Value (TTV).

Since we already classified the session on the Online Classifier and hence we get to know the Correct and the Incorrect Classifications.

Trust Value = Correct Classifications / Total number of sessions classified.

Initially the Trust Value for the Test Session 1 would be set to 1 but after the Test Session arrives at the Online Classifier and had been classified, the New Trust Value is ready to be calculated from the Correct and the incorrect classifications. Also we have a value called the Threshold Test Value (TTV) which is found out to be 0.7 by Trial and Error methods.

Why would we need a Threshold value?. The answer to this would be discussed shortly. Meanwhile, when Test session 1 has been classified by Online Classifier, the Offline classifier do not classify it immediately into Proper and Improper sessions as the Online, Instead it does calculate a probabilistic score for each test session. Remember, Every time each Test Session arrives at the Online Classifier, a new Trust Value is calculated for each session at the Trust Node.

Assume that a Test Session‘t’ arrives at the Online Classifier. The Online Classifier classifies it as a Masquerader or a Proper User. The Trust Node calculates the Trust Value for this Test Session t and supposedly the trust value equals 0.7 which is our TTV. Immediately the control is transferred to the Offline Classifier which generates a Probabilistic score for the Test Session t. In addition to this, it classifies each test session for example Test Session 1 - Test Session t with respect to the Probabilistic Scores for each session. Hence at the t^{th} session the trust value equals the TTV and the Offline Classifier starts classifying each test session as a Proper or a Masquerader. The Model built using an Offline Classifier is more accurate and this Model replaces the Model build by the Online Classifier.

4.2 PROPOSED ALGORITHM

1) The training set for User ‘**u**’ is formed in a text file and this would be our data for forming the proper user model. Furthermore, the training dataset of other Users are prepared which finally contributes to the masquerader model. The complete training set is represented as **R**.
2) Another input for this algorithm would be the streamed sequence **S** of newly unidentified 100 test sessions.

3) An initial classifier model is built for the User ‘**u**’ with the training dataset **R** $\boldsymbol{\theta}^{\wedge\mathbf{0}}$ **=** $(\boldsymbol{\varepsilon}^{\wedge\mathbf{0}}, \mathbf{p}^{\wedge\mathbf{0}}, \mathbf{p}^{\wedge'\mathbf{0}})$**.**

4) The Online and the Offline naïve-bayes algorithm work simultaneously as the test sessions streams in.

5) The online classifier calls a function to generate the unique commands in the current test session and its occurrence rate.

6) The probabilities of the unique commands are taken from training function and are used classify the session as proper or masquerader.

7) Depending on the type of classification we calculate the new probabilities for the commands and update the corresponding vector.

8) Meanwhile, the toggling factor is calculated to allow optimal toggling between the Online and the Offline classifier.

9) If the toggling factor reaches a threshold value, the Offline starts classifying each session from the 1 to the current session when the control was switched to Offline. Let us see how the offline works. Also‘t’ is the session when the control was switched.

10) Now we Iterate through all the 100 test sessions as **t** = 1, 2… 100 or **S** in the sequence.

    a) Initially we start the model as $\boldsymbol{\theta}^{\wedge\mathbf{t}\,=}\,\boldsymbol{\theta}^{\wedge\mathbf{t-1}}$ **.**

    b) Here we try to simulate the working of the offline classifier. As each of the test sessions **t** = 1, 2, …., 100 comes in, we calculate the trust value and say for one particular ‘**t**’, the trust value calculated was not good and hence we repeat these tests for each test sessions until ‘t’, (i, e) 1,2…**t**. Otherwise we continue with step c.

        ✓ Here we use the Expectation-Maximization Algorithm to generate a probabilistic score for these unidentified test session s = 1, 2... t. Well, it has two steps, the E-step and the M-step. In the E-step, the Bayes inversion formula is used to calculate the new scores,

$$P\ (l_s = 1|\ c_{S;}\ \boldsymbol{\theta}^{\wedge\mathbf{t}}),\ \text{for } s = 1, 2...\ t.$$

- ✓ M-step: The model $\boldsymbol{\theta}^{\wedge t}$ **is** updated with the new scores. Hence, at this point the model has information of all sessions s = 1, 2... t and hence is considered more accurate than the Online classifier.

c) E-M step is repeated for different iteration until the conditional variable reaches a threshold vale which should be less than $2.2 * 10^{-11}$.

11) The model at the Online Classifier is updated with this model $\boldsymbol{\theta}^{\wedge t.}$.
12) The algorithm is repeated for different test sessions from step 5.

Finally, we have the model $\boldsymbol{\theta}^{\wedge S}$ for the User '**u**' built on the training set **R** and updated by the sequence **S** of sessions**.**

4.4 EFFECTIVENESS OF BAYESIAN CLASSIFIER

Theoretically, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. Various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains. Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers that do not explicitly use Bayes theorem.

4.5 SIMULATION OF NAÏVE-BAYES ALGORITHM

4.5.1 MACHINE ARCHITECTURE

The Algorithm was developed on Pentium IV processor with 640 MB RAM, 80 GB Hard Disk capacity. The Naïve-Bayes Detection algorithm was simulated using Visual C++ in a .NET environment. Most of the charts were developed using Microsoft Excel.

4.5.2 BRIEF EXPLANATION OF THE ONLINE & OFFLINE DETECTION ALGORITHM

TRAINING PHASE

First thing the algorithm does is to build the training knowledge. It calls the corresponding function to train the proper user dataset and the masquerader dataset. Each of these function collect the unique commands from the 2 datasets and calculates the probability of each command.

Probability of a command ‘c1’ in the proper dataset, $P\ (c_1 \mid u) = nuc1\ _/\ u$

Where ‘n_{uc1}’ is the total number of command ‘c1’ in the proper user dataset u and ‘u’ is the total number of commands in the proper user dataset.

Probability of a command ‘c1’ in the masquerader dataset, $P'\ (c_1 \mid u) = nuc1\ _/\ u$

Where ‘n_{uc1}’ is the total number of command ‘c1’ in the masquerader dataset u and ‘u’ is the total number of commands in the masquerader dataset.

Final Probability with command and CPU Time in the proper dataset,

$$P\ (F \mid u) = P\ (c_1 \mid u)\ *\ P\ (T \mid u)$$

Final Probability with command and CPU Time in the masquerader dataset,

$$P' (F \mid u) = P '(c_1 \mid u) \ * \ P' (T \mid u)$$

4.5.3 TEST PHASE (ONLINE CLASSIFICATION)

Now that we have built the training probabilities of all unique commands from the proper and masquerader dataset, we need to classify the test sessions with these probabilities. The algorithm consists of an input function which reads each of the 100 test sessions. The algorithm uses the Online classifier along with the Offline to classify each test session. The online classifier identifies the unique commands in the current test session and calculates a value called 'temporary' by using the probability (prior probability) of each command obtained from the training knowledge. The temporary value decides if a command is typed by a proper user or masquerader. Let us say a command c1 appears thrice in the current session 's1'.

Temporary value for command c1 in the proper user dataset,

$$T (c_1 \mid s1) \ = \ P (F \mid u) \ * P (F \mid u) * P (F \mid u)\ldots\ldots.(1)$$

$$P (F \mid u) = P (c_1 \mid u) \ * \ P (T \mid u)$$

Where 'P (c_1 | u)' is the probability of that command from the training of proper user data, 'P (T | u)' is the probability of the CPU Time from the training of proper user data. We calculate the maximum obtained CPU time and the probabilities are obtained by scaling other CPU time with reference to the maximum. 'P (F | u)' is the Final probability with both command and CPU time in the proper user data.

Temporary value for command c1 in the masquerader dataset,

$$T '(c_1 \mid s1) \ = \ P' (F \mid u) \ * P '(F \mid u) * P '(F \mid u)\ldots\ldots(2).$$

$$P' (F \mid u) = P '(c_1 \mid u) \ * \ P' (T \mid u)$$

Where 'P' (c_1 | u)' is the probability of that command from the training of masquerader data, 'P' (T | u)' is the probability of the CPU Time from the training of masquerader data and 'P' (F | u)' is the Final probability with both command and CPU time in the Masquerader data.

If the temporary value for a command c1 in the proper user dataset is greater than the temporary value in the masquerader dataset we classify that command to be a proper user command otherwise it is classified as a masquerader. Hence these comparisons are repeated with all unique commands in the current test session. If the majority of the commands are proper we declare the test session as proper and masquerader otherwise.

The temporary value for a command is not the new probability of the command in the current session.

New probability of the command c1 for the current session s1 in proper user dataset,

$$P\ (c_1 \mid u)\ = (n_{uc1} + n_{s1c1)}\ /\ (T_u + T_{s1})$$

Where 'n_{uc1}' is the number of occurrence of command c1 in the proper user dataset, 'n_{s1c1}' is the number of occurrence of command c1 in the current session s1, 'T_u ' is the total number of commands in the Proper user dataset and 'T_{s1}' is the total number of commands in the current session s1. This new probability is multiplied with new probability of CPU time to get the Final probability for the proper user data.

New probability of the command c1 for the current session s1 in masquerader dataset,

$$P'\ (c_1 \mid u)\ =\ (n_{uc1} + n_{s1c1)}\ /\ (T_u + T_{s1})$$

Where 'n_{uc1}' is the number of occurrence of command c1 in the masquerader dataset, 'n_{s1c1}' is the number of occurrence of command c1 in the current session s1, 'T_u' is the total number of commands in the masquerader dataset and 'T_{s1}' is the total number of commands in the current session s1. This new probability is multiplied with new probability of CPU time to get the Final probability for the masquerader data. The probability P (c_1 | u) & P' (c_1 | u) are substituted in the above equation 1 & 2 to get the Total probability.

At this time, update the new probabilities of each command with the training probabilities in the corresponding vector. In our simulation program, we had two dynamic vectors which stored the training probabilities of each commands in the proper and masquerader dataset. Hence if the current session is a proper one, we update the Proper vector otherwise we update the masquerader vector.

The Toggling Factor is a value which decides to switch on the control to the Offline classifier.

4.5.3.1 Toggling Factor

This Factor plays a very significant role in the accuracy of the whole system. In our detection system we have the Online and the Offline working simultaneously. Moreover, we have a toggling factor which is calculated every time a new session is read. Why do we say the toggling factor is important? The Online classifier classifies each command with the probabilities determined from the Training function. The training function calculates the probability of each command appearing in the training dataset. If the Online does not find a probability for a command from the training function it assumes a small value **'α'** that is initialized to 0.01. Hence the knowledge or the classification results of the Online classifier may or may not be accurate. Now, let us see what the Offline classifier does. The Offline classifier comes into action only when the toggling factor is true and let us say that it was true at the 't^{th}' test session. In fact this classifier generates the probabilistic score for each session by considering all the cumulative probabilities for a command appearing in each session from 1 to t. The Expectation-Maximization algorithm (EM) is used to calculate the probabilistic score for each session. The offline classifier when compared with the online is thus more accurate and trustworthy. Apparently, the probability of each command calculated by the Offline is also accurate. However, the Offline classifier cannot be used completely to classify all the test sessions since it can be too pricy with respect to the time consumed by the classifier. Hence it has to be used in combination with the Online. Furthermore, we could make the probabilities calculated by the offline available for the Online to classify the rest of the test sessions and thus we would save some time consumed by the whole algorithm. It is therefore clear that the Offline cannot be used for every session which comes in to the detection system. However, if we do not use the offline at the right time we would not be able to improve the accuracy on the Online classifier. This right time is decided by the Toggling Factor. This factor is calculated by repeated experimentation of the algorithm. From our dataset, we have decided that the Toggling Factor is the session number which is a multiple of 3. Hence for every session which is a multiple of 3 the offline function is called and the probabilities of different commands calculated by the offline are updated into the Online. From here the online classifies each session based on the updated probability.

The toggling factor was chosen based on the accuracy result of the detection system. When the toggling factor was the session number with the multiple of 4, we had a hit rate of 80%. The hit rate decreased to 77% when the factor was set as a multiple of 5. Similarly, hit rates were calculated for several toggling factors. The factor which gave the best results were chosen as the Toggling factor for the detection system and in our case toggling factor which was the multiple of 3 gave the optimal results and hence chosen.

4.5.3.2 Scaling Factor or Threshold Value

The Threshold Value or the Scaling Factor plays an important role in the Offline part of the Detection System. Probabilistic score for each test session is being assigned by the E-M Algorithm. The E-M algorithm has two steps involved in it. The first step is the Expectation mechanism where we calculate a value named as ’l_s’. The variable ’l_s’ is known as the

Indicator variable. The indicator variable stores a probabilistic value for a command which is the sum of the prior probabilistic value of that command in the proper user dataset and the masquerader user dataset. The variable is updated with a new value after the Maximization step for each iteration. After calculation of the indicator variable in the Expectation step we now use the Maximization step to calculate the final Probabilistic score for the command. Here we use all the information for that command by considering all the test sessions from 1 to t. This new value becomes the value that needs to be updated to the Indicator variable. Thus for a command we use the expectation step and the maximization step to calculate the final score and thus forming the first iteration. The iteration for that command is continued until we have reached an optimal score for that command. The final score of that command in the first iteration is used to calculate the indicator variable 'l_s' (Expectation step) of the command in the second iteration. Using this indicator variable we calculate the probabilistic score of the command in the Maximization step. This iteration is repeated until we reach our desired scores for that command. This desired score is achieved only when the loop exits by meeting the scaling factor. In other words, the iteration is continued until the condition variable reaches a particular value. With our dataset, the particular value which is otherwise known as the Scaling factor is calculated as $2.2 * 10^{-11}$. We now describe how we calculate the condition variable. The absolute value of the command in the proper dataset is calculated by finding the difference between the Initial and the final indicator variable value for that iteration. The absolute value of the same command in the masquerader dataset is calculated by finding the difference between the Initial and final Indicator variable for the same iteration. The condition variable is nothing but the sum of the squares of the absolute values for that command in the proper and masquerader dataset. We did run a number of iterations for different users to calculate this value and based on the accuracy results of these experiments we gave the scaling factor a value of $2.2 * 10^{-11}$. Moreover, using this factor value we had the best accuracy results in terms of Hit rate, Missing alarms and False Alarms.

4.5.4 TEST PHASE (OFFLINE CLASSIFICATION)

The working of the offline classification is explained briefly in the following subsection. As each test session comes in at the input function the offline classifier calculates a cumulative probability score using the E-M Algorithm (Expectation Maximization). The Expectation is nothing but the probability of the command determined by the Bayes inversion formula.

$$P(1_s = 1|c_s;\theta^t) = \frac{P(1_s = 1;\theta^t)P(c_s|1_s = 1;\theta^t)}{P(c_s;\theta^t)},$$

where

$$\begin{aligned} P(c_s;\theta^t) &= P(1_s = 0;\theta^t)P(c_s|1_s = 0;\theta^t) + \\ &\quad P(1_s = 1;\theta^t)P(c_s|1_s = 1;\theta^t). \end{aligned} \tag{7}$$

Also $P(c_s|1_s = 0;\theta^t)$ is the cumulative prior probability of the command in the proper user dataset and $P(c_s|1_s = 1;\theta^t)$ is the cumulative prior probability of the command in the masquerader dataset. For the first iteration the training probabilities are used to initialize the cumulative prior probability vector. The new cumulative probability $P(1_s = 1|c_s;\theta^t)$ is stored in a variable. Next, we start with the Maximization step,

The final probabilistic score for a command in the proper user dataset is,

$$\hat{p}_c^t \;=\; \frac{\alpha_c - 1 + n_{+c}^0 + n_{+c}^t}{\sum_{v=1}^{C} (\alpha_v - 1 + n_{+v}^0 + n_{+v}^t)} \qquad (7)$$

where α_c is a small value assumed to be1.01,

α_v is nothing but α_c ,

n_{+c}^0 denotes the total count of command c among proper sessions in the training set,

n_{+v}^0 denotes the total count of all commands (1 to v) among proper sessions in the training set,

$$n_{+c}^t \;=\; \sum_{s=1}^{t} (1 - 1_s) n_{sc}$$

where,

1_s is the indicator variable which stores the cumulative probability value,

n_{sc} is the count of command c in the current session

n_{+v}^t is nothing but n_{+c}^t value calculated for commands 1 to v.

The above probability is updated with the command's prior probability in a dynamic vector which is named as the cumulative probability vector for proper user dataset.

The final probabilistic score for a command in the masquerader dataset is,

$$\hat{p}'^{t}_{c} = \frac{\alpha'_{c} - 1 + n'^{0}_{+c} + n'^{t}_{+c}}{\sum_{v=1}^{C} \left(\alpha'_{v} - 1 + n'^{0}_{+v} + n'^{t}_{+v}\right)} \tag{7}$$

where α'_{c} is a small value assumed to be 1.01,

α'_{v} is nothing but α_{c} ,

n'^{0}_{+c} denotes the total count of command c among masquerader sessions in the training set,

n'^{0}_{+v} denotes the total count of all commands (1 to v) among masquerader sessions in the training set,

$$n'^{t}_{+c} = \sum_{s=1}^{t} 1_{s} n_{sc}$$

where,

1_{s} is the indicator variable which stores the cumulative probability value,

n_{sc} is the count of command c in the current session

n'^{t}_{+v} is nothing but n'^{t}_{+c} value calculated for commands 1 to v.

The above probability is updated with the command's prior probability in a dynamic vector which is named as the cumulative probability vector for masquerader dataset.

The above E-M algorithm is repeated for different iteration until the scaling factor condition for that command is reached. The Final probability (P (F | u), P' (F | u)) is obtained from the product of the probabilities of CPU Time and the command in a session. Finally, the probabilities are updated in the cumulative probability vector.

P (c_1 | u) = $\hat{p}^{t}_{c}$

P (F | u) = P (c_1 | u) * P (T | u)

P' (c_1 | u) = $\hat{p}'^{t}_{c}$

$$P' (F \mid u) = P' (c_1 \mid u) * P' (T \mid u)$$

Now we have all the new probabilities calculated by the Offline classifier and we begin to classify each session.

The Offline classification is done with a group of test sessions say from session 1 to session t (when the toggling factor condition is true). The temporary value is calculated in the Offline classifier just like we did in the online classifier. Each test session from 1 to t is classified against these new probabilities. Finally we update the new probabilities in the main Vector for the Online classifier to classify the rest of the sessions. This improves the accuracy of the whole system.

4.6 EXPERIMENTAL RESULTS

The dataset was trained and tested using the above proposed algorithm. A brief discussion on their accuracy and timing performance is given below.

4.6.1 ACCURACY

4.6.1.1 Original Classification

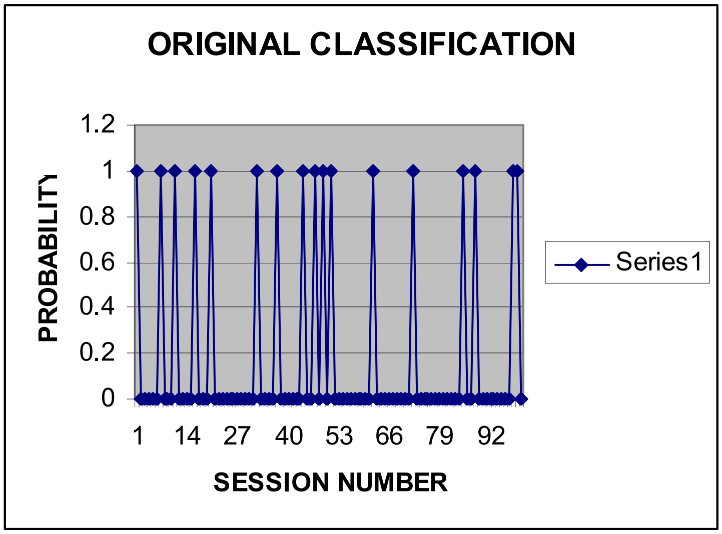


Figure 5 original classification for USER 1

This classification is the truth which we already know in our case since we have the test data. The test sessions from 1 to 100 are taken on the X-AXIS and the probability from 0 to 1 are taken on the Y-AXIS. All the test sessions hitting the probability 1 are masquerade test sessions. Hence these analyses are based on the original truth and not results from the algorithm.

4.6.1.2 Results from classifiers for different users

The average of results of the Online & Offline Naïve-Bayes Classifier from User dataset 1, 3, 5 and 9 are shown below.

Online Classifier with Commands only

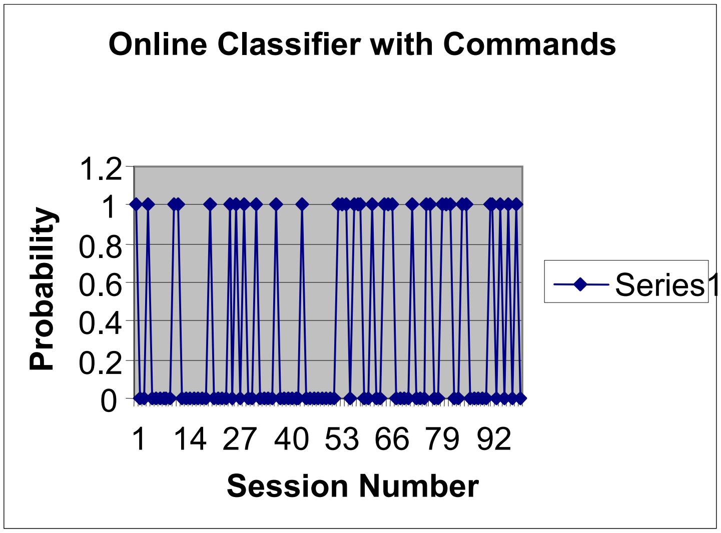


Figure 6 Online Classifier with Commands only

Here we again have the test sessions in the X-axis and the probability on the Y-axis. The analysis is based on the User dataset 1. The Online classifier had a Hit rate of 70%. But it had a high false alarm rate of 24% and a 6% missing alarm rate.

Online Classifier with Commands and CPU Time

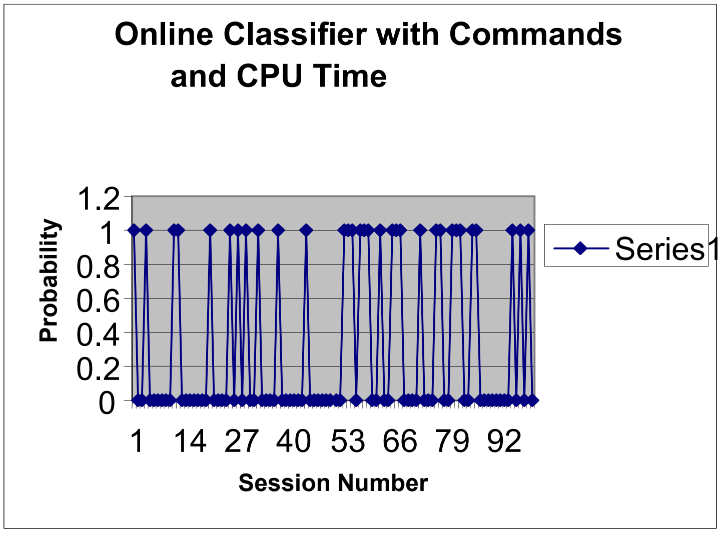


Figure 7 Online Classifier with Commands and CPU Time

The above figure shows us the chart of the simulation results for the Naïve-Bayes Online classifier with Commands and CPU time. The chart shows us the classification of proper and masquerade sessions. Hit rate with this classifier was 72 %, the false alarm rate reduced by 2 %. Hence the performance of the classifier improved in terms of false alarm rate and hit rate.

Online & Offline Classifier with Commands only

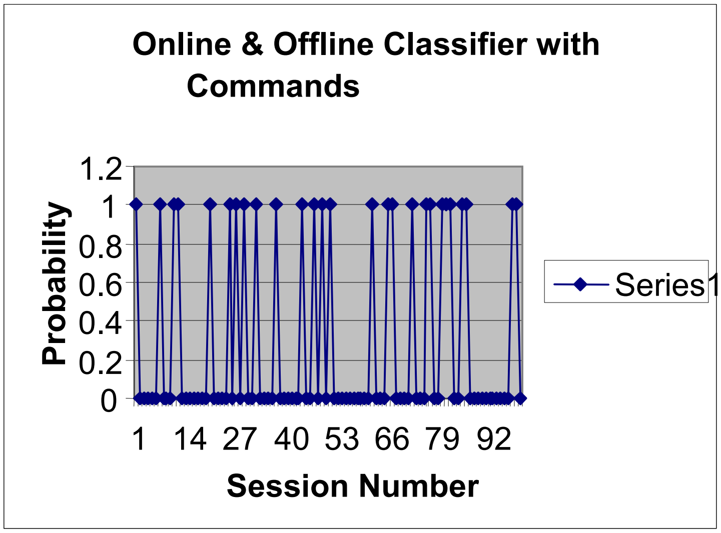


Figure 8 Online & Offline Classifier with Commands only

Now we allow the test sessions to be classified with the Naïve-Bayes Online and Offline classifier. The results have been expressed in the form of a chart as shown in figure. Again we have the test sessions on the X-axis and the Probability on the Y-axis. When compared with the Online classifier, we had better results with a hit rate of 85% and the False alarm rate reduced by 12%. Moreover, the recorded missing alarm rate was 3%. The overall performance was good with this classifier.

Online & Offline Classifier with Commands and CPU Time

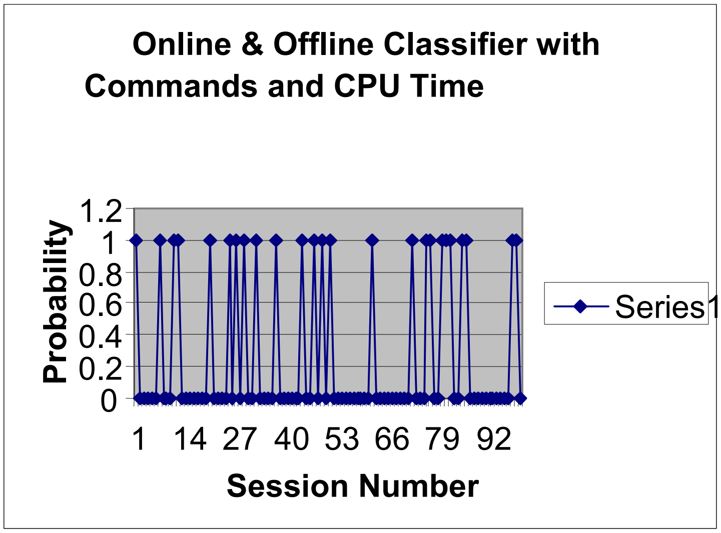


Figure 9 Online & Offline Classifier with Commands and CPU Time

During the implementation of this classifier, we considered both the commands and the CPU time. The results have been depicted in the form a graph as shown in the figure. Session numbers are on the X-axis and the corresponding probabilities were taken on the Y-axis. The hit rate raised to 88% while the false alarm decreased by 3% and the missing alarm rate recorded a low of 3%. Comparatively, the performance of the Online & Offline Naïve-Bayes classifier performed better when we considered both commands and the CPU time.

USER 1

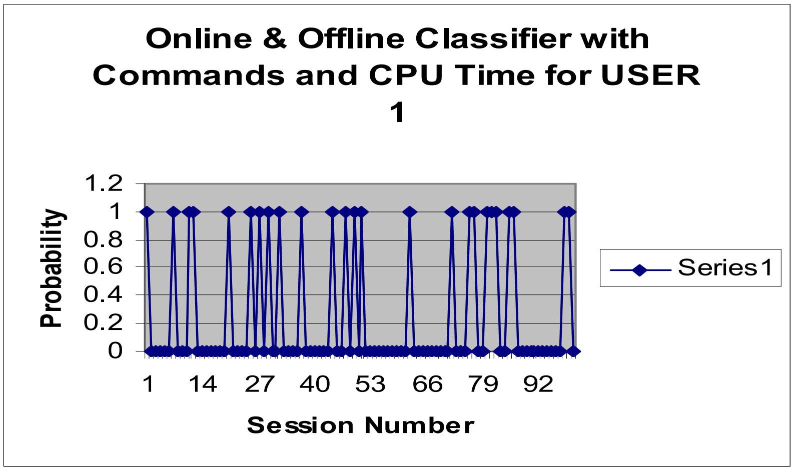


Figure 10 Online & Offline classifier with command and CPU time for USER 1

The results for USER 1 have been shown in the above figure. The online & offline Naïve-Bayes classifier with commands and time was used to generate the results. A hit rate of 88% was recorded with a false alarm rate of 9% and a missing alarm rate of 3%.

USER 3

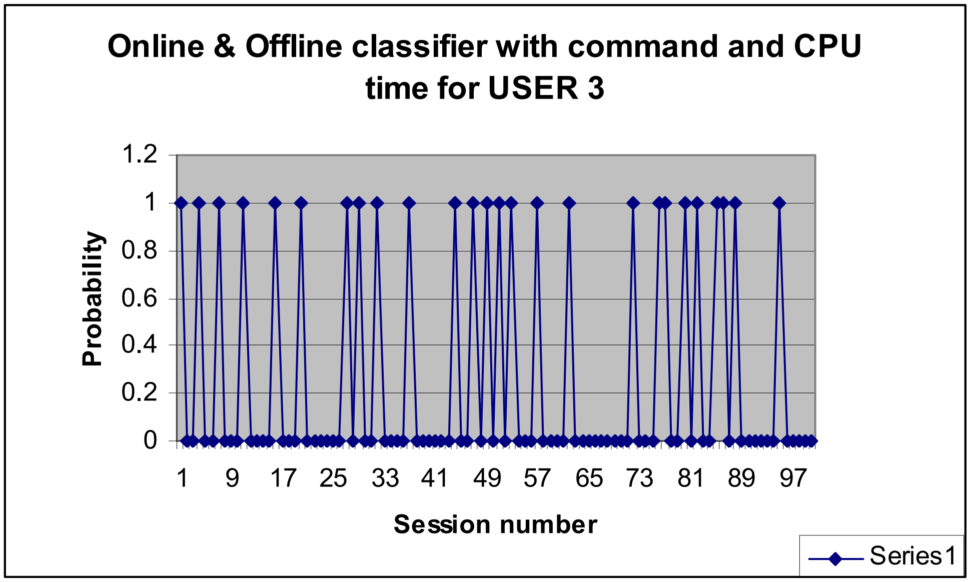


Figure 11 Online & Offline classifier with command and CPU time for USER 3

The results for USER 3 have been shown in the above figure. The online & offline Naïve-Bayes classifier with commands and time was used to generate the results. A hit rate of 87% was recorded with a false alarm rate of 11% and a missing alarm rate of 2%.

USER 5

The results for USER 5 have been shown in the above figure. The online & offline Naïve-Bayes classifier with commands and time was used to generate the results. A hit rate of 90% was recorded with a false alarm rate of 5% and a missing alarm rate of 5%.

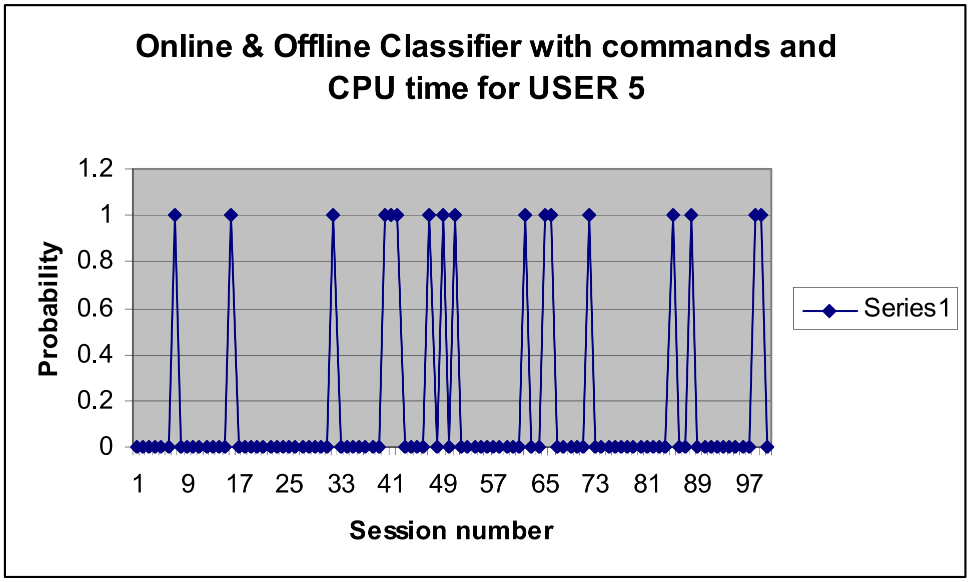


Figure 12 Online & Offline classifier with command and CPU time for USER 5

USER9

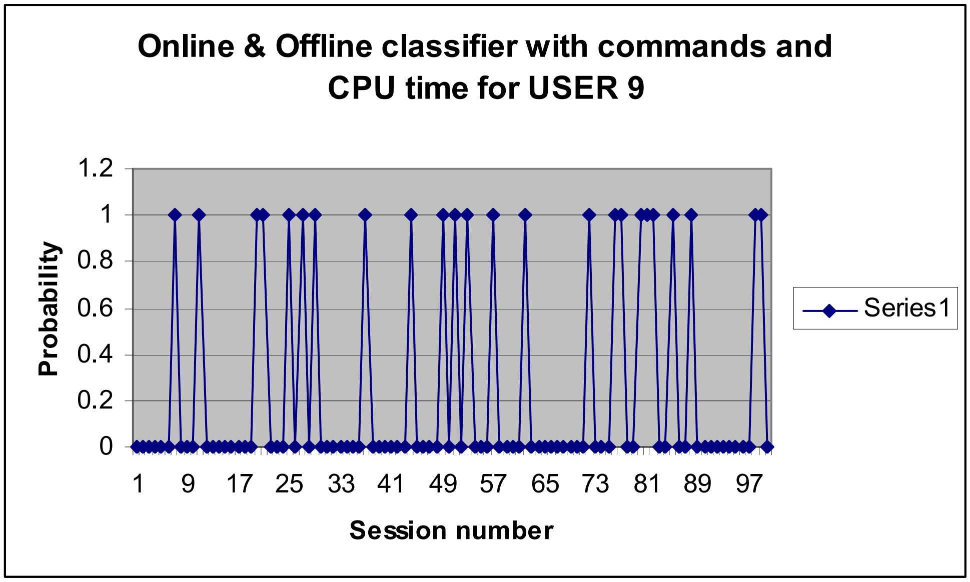


Figure 13 Online & Offline classifier with command and CPU time for USER 9

The results for USER 9 have been shown in the above figure. The online & offline Naïve-Bayes classifier with commands and time was used to generate the results. A hit rate of 87% was recorded with a false alarm rate of 10% and a missing alarm rate of 3%.

4.6.1.3 ROC Curves

Each user was trained and tested with the Online & Offline with commands only classifier and Online & Offline with commands & time classifier respectively. To evaluate the method's success over the entire dataset, a ROC curve could be very useful. Here we have the false alarm rate on the X-axis and the Hit rate on the Y-axis.

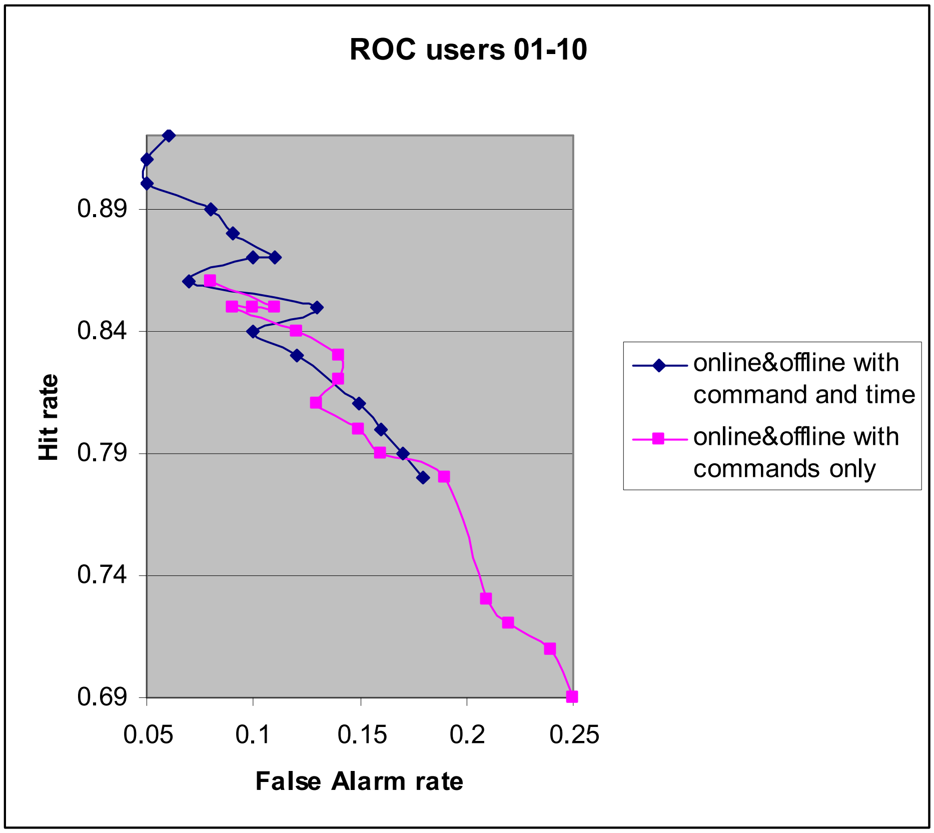


Figure 14 ROC Curve

From the above chart we infer that the online & offline with command and time classifier clearly out performs the online & offline with command only classifier.

4.6.2 TIMING PERFORMANCE

4.6.2.1 Online Classifier with Commands and CPU Time

The online classifier classifies a test session immediately. If the session is proper the corresponding vector is updated and if it is masquerader the corresponding vector is updated. Experiments were conducted to find the time complexities between different classifiers. Each session going through the online classifier with commands and CPU time took the least amount of time when compared to the sessions classified through other classifiers.

Time taken to classify one session in the Online Classifier = 0.05 secs

Time taken to classify 100 test sessions in the Online classifier = 5 secs

4.6.2.2 Offline Classifier with Commands and CPU Time

The offline classifier classifies each session in a cumulative basis and there is no Toggling factor involved with this classification. After the 100 th session has arrived the classifier would have given the most accurate result. Each session going through the Offline Classifier with commands and CPU Time took the most amount of time when compared to the sessions classified through other classifiers. The time complexities are shown below.

Time taken to classify one session in the Offline classifier = 2.3 secs

Time taken to classify 100 test sessions in the Offline classifier = 230 secs

4.6.2.3 Online & Offline Classifier with Commands and CPU Time

In the Online & Offline Classifier with Commands and CPU Time, the Online classifier uses the model or probabilities updated by the offline classifier to classify the rest of the sessions. The control to the Offline classifier is given by the Toggling Factor. This classifier was most effective when compared to the time and detection rate.

Time taken to classify one session in the Online & Offline classifier = 1.3 secs

Time taken to classify 100 test session in the Online & Offline classifier = 130 secs

Accuracy results & timing performance for different classifiers

Classifiers	Hit Rate (HR) %	False Alarm rate (FAR) %	Missing Alarm rate (MAR) %	Timing Performance per session (sec)
Online	72	22	6	0.05
Online & Offline	88	9	3	1.3
Offline	94	5	1	2.3

Table 3

CHAPTER V

CONCLUSION

The Online and the Offline Bayesian classifier is more accurate than the Simple Naïve Bayes Classifier. A comparison result of the accuracy rate of the Online and Offline Bayesian classifier with that of the Simple Bayes Classifier shows that Online & Offline classifier had better accuracy results in terms of Hit rate, Missing Alarm rate and False alarm rate. Moreover, the addition of another parameter namely the "CPU Time" increased the overall accuracy of the Model. We compared the Online & Offline classifier with "Command" as the only parameter against the classifier with both the "Command" and the "CPU Time" and found that the later had an accuracy rate of 88%. Most of the accuracy results have been shown in Table 3. Finally, in assessing the results of a Masquerader Detection System we would be really concerned with the trade-off between correct detections (hits or true positives) and false detections (false alarms, or false positives) of the different classifiers. Therefore it is often depicted on a receiver operating characteristic curve (called the ROC curve) where the percentages of hits and the false alarms are shown on the Y-axis and the X-axis, respectively. From the ROC curve we infer that the Online & Offline classifier with commands and CPU Time outperformed the classifier with commands by a significant margin.

5.1 FUTURE WORK

Integration of a different algorithm other than the Naïve Bayes classifier with the Online and the Offline techniques can be one of the potential future researches. Also the Expectation – Maximization algorithm which uses the log likelihood to generate a probabilistic score can be replaced by other models. Addition of different parameters in the dataset could increase the accuracy of the model and thereby increasing the levels of surveillance. Finally, one may want to build a statistical learning model to calculate the optimum trust value used to toggle around the Online and the Offline classifier.

REFERENCES

[1] Roy A. Maxion and Tahlia N. Townsend. “Masquerade Detection Using Truncated Command Lines.” *International Conference on Dependable Systems and Networks (DSN-02)*, pp. 219-228, Washington, DC, 23-26 June 2002. IEEE Computer Society Press, Los Alamitos, California, 2002.

[2] Matthias Schonlau, William DuMouchel, Wen-Hua Ju, Alan F. Karr, Martin Theus, and Yehuda Vardi. “Computer intrusion: detecting masquerades. Statistical Science, 16(1):58-74, February 2001.

[3] W. DuMouchel. Computer intrusion detection based on Bayes factors for comparing command transition probabilities. Technical Report 91, National Institute of Statistical Sciences, Research Triangle Park, North Carolina 27709-4006, 1999.

[4] W. Ju and Y.Vardi. A Hybrid high-order markov chain model for computer intrusion detection. Technical Report 1992, National Institute of Statistical Sciences, Research Triangle Park, North Carolina 27709-4006, 1999.

[5] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for Anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295-331, August 1999.

[6] B.D. Davison and H. Hirsh. Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time Series Problems,* papers from the 1998 AAAI Workshop, 27 July 1998, Madison, Wisconsin, pages 5-12. Published as AAAI Technical Report WS-98-07, AAAI Press, Menlo Park, California, 1998.

[7] Yung, Kwong h. “Using Self-Consistent Naïve Bayes to detect Masquerades”, Stanford ECJ, spring 2004.

[8] Domingos, Pedro, Pazzani, Michael. “Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier”.

[9] Mitchell, Tom M. “Generative and Discriminative Classifiers: Naïve Bayes and

Logistic regression", DRAFT OF September 5, 2005.

[10] Arthur P. Dempster, Nam M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM Algorithm." Journal of the Royal Statistical Society, Series B, 39(1), 1-38, 1977.

[11] Han, Jiawei. Kamber, Micheline. "Data Mining Concepts and techniques". Simon Fraser University, Morgan Kaufmann publishers. *An Imprint of Elsevier Science.* San Francisco, CA.

VITA

Toni Kunnel

Candidate for the Degree of

Master of Science

Thesis: AN OPTIMIZED NAÏVE-BAYES DETECTION SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Mavelikara, Kerala, India, on April 7, 1980, the son of Mr. K. M. Varughese and Mrs. Rebeccamma Varughese.

Education: Obtained Senior High School Diploma from S. V. M, India in May 1997. Received Bachelor of Engineering in Computer Science & Engineering from Bharathiyar University, Coimbatore, India in October 2001. Completed the requirements for the Master of Science Degree at Oklahoma State University in December 2005.

Experience: March 2002 – January 2003. Worked as a software developer for Hinduja Telecomm. December 2003 – December 2005. Part-time Operator for the Information Technology Division of Oklahoma State University.

ABSTRACT

Name: Toni Kunnel Date of Degree: December 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: An Optimized Naïve-Bayes Detection System

Pages in Study: 53 Candidate for the Degree of Master of Science

Major Field: Computer Science

Abstract:

A Masquerader is a malicious user who tries to gain access or control of a system from a proper user. The objective of this thesis is to increase the accuracy of the existing Naïve-Bayes Algorithm for detecting Masquerade attempts. We have an Online and an Offline classifier. The Classifier used in our experiments is the Naïve-Bayes Classifier. Although the dataset is being learned by the Online and the Offline classifier simultaneously, the online classifier makes an instantaneous decision whereas the Offline makes it after a specified span of time.

We try to increase the accuracy of the detection system by increasing the number of parameters within the dataset and also by the introduction of a Toggling factor between the Online and the Offline classifiers. The Naïve-Bayes classifier builds a proper user model and an improper model from the training dataset. The Test sessions are classified against these models. The E-M Algorithm was used to generate a probabilistic score for the unidentified sessions in the testing phase. The dataset was prepared from the log files of different users that logged into the Computer Science Administrative Server (a.cs.okstate.edu) for Oklahoma State University. Experimental results demonstrate that the Online & Offline classifier with commands and the extra parameter namely the CPU time outperformed the Online & Offline classifier with commands in terms of both the false alarm rate and the hit rate.

Advisor's Approval: Dr. Johnson P Thomas